flowQB – Automated Quadratic Characterization of Flow Cytometer Instrument Sensitivity*

Josef Spidlen[1], Wayne Moore[2], Faysal El Khettabi,
David Parks[2], Ryan Brinkman[1]

[1] *Terry Fox Laboratory, British Columbia Cancer Agency*
Vancouver, BC, Canada

[2] *Genetics Department, Stanford University School of Medicine*
Stanford, California, USA

August 31, 2016

1 Abstract

This package provides methods to calculate flow cytometer's detection efficiency (Q) and background illumination (B) by analyzing LED pulses and multi-level bead sets. The method improves on previous formulations **???** by fitting a full quadratic model with appropriate weighting, and by providing standard errors and peak residuals as well as the fitted parameters themselves. K-means clustering is incorporated for automated peak detection of multi-peak data. Overall, this method is successful in providing good estimates of the Spe scales and backgrounds for all of the fluorescence channels on instruments with good linearity. Known photoelectron scales and measurement channel backgrounds make it possible to estimate the precision of measurements at different signal levels and the effects of compensated spectral overlap on measurement quality.

2 Get the Latest Version

Examples in this vignette, documentation and unit tests are mostly dependent on data that is available in the flowQBData library. This library is available as of BioConductor 3.4. You are looking at a vignetter based on BioConductor 3.3. Therefore, we had to comment out most executable examples in order for this to compile. Please install flowQBData manually if you want to run those examples. Or even better, please use the latest devel version of flowQB, which is available from `http://bioconductor.org/packages/devel/bioc/html/flowQB.html`. That way, all the examples etc. will work properly.

*This project was supported by the Terry Fox Foundation, the Terry Fox Research Institute and the Canadian Cancer Society. Josef Spidlen is an ISAC Marylou Ingram Scholar.

3 Introduction

The basic measurement capabilities of a fluorescence cytometer measurement channel can be estimated by knowing just two values: Q, the statistical number of photoelectrons (Spe) generated per unit of dye in the sample, and B, the background light in dye equivalents that sets the minimum variance that underlies all measurements. From these the minimum detectable amount of dye and the precision of measurements at different signal levels can be calculated. Therefore, measurements of Q and B are the key to making valid comparisons between different instruments and among different channels on an instrument.

We can achieve full specification of Q and B in two steps by evaluating photoelectron scales for the measurement channels of interest and relating those to measurements on dye reference samples. There are problems and challenges in implementing each step in a way that is convenient and accurate in routine use. Here we address the photoelectron scale aspect by providing reliable automated software for obtaining the needed information from LED or multi-level, multi-dye bead data.

When a cell or particle moves through a sensing area on a flow cytometer, a focused laser beam excites fluorescent dyes that emit a pulse of light in all directions. The optical system collects some of this light and passes it through optical filters to the cathodes of one or more PMT detectors where a fraction of the photons generate photoelectrons. The photoelectrons are amplified through a series of dynodes to produce an output current pulse. The photoelectron production is a Poisson process, so the variance of signals at a particular level is proportional to the signal level itself. Since the amplification process through the dynodes introduces some additional variance, we introduce the term "statistical photoelectrons" or Spe to denote the effective Poisson number relating signal levels and their variances.

Since it is difficult to measure photoelectron counts directly, the usual method for estimating photoelectron scales has been to measure uniform light signals at various levels and fit the measured means and variances to a statistical model involving the Poisson distribution expectations for the relation between them. Historically, two kinds of signal source have been used: an LED light source producing very uniform pulses at adjustable signal levels or microspheres (also called "beads") labeled with several different concentrations of a mixture of fluorescent dyes. The model can be expressed as a second-degree (quadratic) polynomial relating the observed signal means and variances to Q, B and, for microsphere samples, CV_0, a common non-statistical variability in the measurements of the different microsphere peaks.

For measurements on a population of similar particles like one of the populations in a multi-level bead set, the observed variance should be the sum of the electronic noise, background light not associated with the particles, Poisson variance related to the signal levels of the particles, the variation in dye amount among the particles and illumination variations due to particles taking different flow paths through the laser beam. The electronic noise and background light combine into a variance contribution that is not dependent on the signal level. The Poisson variance is proportional to the signal level, and the dye and illumination variations combine to form CV_0 whose contribution to the variances goes with the square of the signal level. Therefore, the general equation for the observed variance can be expressed as

$$V(M_I) = c_0 + c_1 M_I + c_2 M_I^2$$

where M_I is the mean signal of the population in measurement intensity units, and $V(M_I)$ is

the observed variance of a population with mean M_I. Methods implemented in this package fit sets of mean and variance data points with this equation to obtain the parameters c_0, c_1 and c_2. If we scale the signal levels in Spe, we can define B_{Spe} as the effective background level in Spe and M_{Spe} as the mean signal in Spe. Then, defining Q_I as the Spe per intensity unit, as explained in our paper **?**, we can calculate

$$B_{\mathrm{Spe}} = c_0/c_1^2$$

$$Q_I = 1/c_1$$

$$CV_0^2 = c_2$$

4 Requirements

The flowQB package requires the flowCore library in order to read FCS files **?**, the stats package for its model fitting and also K-means clustering functionality, the extremevalues package to determine outliers, and it also makes use of some basic functions and classes from the methods and BiocGenerics packages. In addition, we suggest installing the flowQBData package, which contains example data sets to demonstrate the functionality of this package. Alternatively, one can use the FlowRepositoryR package in order to work directly with data stored in FlowRepository. In addition, we recommend the xlsx package in order to parse house-keeping information from MS Excel spreadsheets. Finally, we use the RUnit package in order to implement unit tests assuring consistency and reproducibility of the flowQB functionality over time. Let's start by loading the flowQB package as well as the flowQBData package with data to demonstrate the functionality of this package.

```
> library("flowQB")
> ## library("flowQBData")
> ## flowQBData is available since BioConductor 3.4, please install it
> ## manually in order to be able to run examples in this vignette
```

5 Fitting LED pulser data

An LED light pulser is producing very uniform pulses at adjustable signal levels. White LEDs provide some signal at all visible wavelengths, but the far-red emission is weak. A given LED pulse level will generate quite different photoelectron signals on different detectors, so it is important to collect data over a wide range of LED levels to assure that the measurement series on each detector will include the low, middle and high level signals needed for optimal results in the fitting procedure. The `fit_led` function assumes that data generated by different LED levels are provided as separate FCS files. These files are passed to the `fit_led` function in the form of a vector of FCS file paths. In addition, house keeping details about the data and the way the fitting procedure should be performed need to be provided, resulting in the following list of arguments:

- `fcs_file_path_list` – A vector of FCS file paths pointing to data generated by an LED pulser set to a range of LED levels; different levels generated different FCS files, all data coming from a single instrument.

- `ignore_channels` – A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

- `dyes` – A vector of dye names that you would normally use with the detectors specified below. This value does not affect the fitting, but those dyes will be "highlighted" in the provided results.

- `detectors` – A vector of short channel names (values of the $PnN keywords) specifying channels matching to the dyes specified above. The length of this vector shall correspond to the length of the dyes vector. These channels should be all of the same type as specified by the `signal_type` below, i.e., area, height or width of the measured signal.

- `signal_type` – The type of the signal specified as the "area", "height" or "width". This should match to the signal type that is being captured by the channels specified in the detectors argument. The signal type is being used in order to trigger type-specific peak validity checks. Currently, if signal type equals to "height" then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. In addition, peaks that are not sufficiently narrow (*i.e.*, exceeding a specific maximum CV) are also omitted from the fitting. Currently, the maximum allowed CV is set to 0.65, but the code is designed to make this user-configurable and signal type dependent eventually.

- `instrument_name` – The make/model of the instrument. The purpose if this argument is to allow for instrument-specific peak validity checks. At this point, if "BD Accuri" is passed as the instrument type, then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. Additional instrument-specific peak validity checks may be implemented in the future.

- `bounds` – On some instruments, the lowest LED peaks may be cut off at a data baseline so that the peak statistics will not be valid. Therefore, peaks too close to the baseline need to be excluded from the fitting. Also, many instruments do not maintain good linearity to the full top of scale, so it is also important to specify a maximum level for good linearity and, on each fluorescence channel, exclude any peak that is above that maximum. The bounds argument shall provide a list specifying the minimum and maximum value for the means of valid peaks; peaks with means outsize of this range will be ignored for that particular channel.

- `minimum_useful_peaks` – Different peaks may be omitted for different channels due to various validity checks described above. This argument specifies the minimal number of valid peaks required in order for the fitting procedure to be performed on a particular fluorescence channel. Generally, fitting the three quadratic parameters requires three valid points to obtain a fit at all, and 4 or more points are needed to obtain error estimates. Requiring higher values would exclude some of your data but likely produce better results.

- `max_iterations` – The peaks have a wide range of variances, so unweighted least squares fitting is not appropriate, and we need to apply appropriate weights in the fitting proce-

dure. In particular, the populations with lower variances get more weight since having the fit miss them by any particular amount is worse than missing a high variance population by the same amount. This argument specifies the maximum number of iterations for the iterative fitting approach with appropriate weight recalculations. In most cases, the fitting converges relatively fast. The iterating stops when either the maximum of iterations is used or if none of the coefficients of the model changed more than 0.00005. The default maximum of 10 iterations seems to be enough in most cases. You can also explore your results in order to see how many iterations were actually done for each of the all of the fitting.

```
> ## Example is based on LED data from the flowQBData package
> fcs_directory <- system.file("extdata", "SSFF_LSRII", "LED_Series",
+     package="flowQBData")
> fcs_file_path_list <- list.files(fcs_directory, "*.fcs", full.names= TRUE)
> ## We are working with these FCS files:
> basename(fcs_file_path_list)
> ## Various house keeping information
> ## - Which channels should be ignored, typically the non-fluorescence
> ##    channels, such as the time and the scatter channels
> ignore_channels <- c("Time",
+     "FSC-A", "FSC-W", "FSC-H",
+     "SSC-A", "SSC-W", "SSC-H")
> ## - Which dyes would you typically use with the detectors
> dyes <- c("APC", "APC-Cy7", "APC-H7", "FITC", "PE", "PE-Cy7", "PerCP",
+     "PerCP-Cy55", "V450", "V500-C")
> ## - What are the corresponding detectors, provide a vector of short channel
> ## names, i.e., values of the $PnN FCS keywords.
> detectors <- c("APC-A", "APC-Cy7-A", "APC-Cy7-A", "FITC-A", "PE-A", "PE-Cy7-A",
+     "PerCP-Cy5-5-A", "PerCP-Cy5-5-A", "Pacific Blue-A", "Aqua Amine-A")
> ## - The signal type that you are looking at (Area, Height or Width)
> signal_type <- "Area"
> ## - The instrument make/model
> instrument_name <- 'LSRII'
> ## - Set the minimum and maximum values, peaks with mean outsize of this range
> ## will be ignored
> bounds <- list(minimum = -100, maximum = 100000)
> ## - The minimum number of usable peaks (represented by different FCS files
> ## in case of an LED pulser) required in order for a fluorescence channel
> ## to be included in the fitting. Peaks with mean expression outside of the
> ## bounds specified above are omitted and therefore not considered useful.
> ## Fitting the three quadratic parameters requires three valid points to obtain
> ## a fit at all, and 4 or more points are needed to obtain error estimates.
> minimum_fcs_files <- 3
> ## - What is the maximum number of iterations for iterative fitting with
> ## weight adjustments
```

```
> max_iterations <- 10 # The default 10 seems to be enough in typical cases
> ## Now, let's calculate the fitting
> led_results <- fit_led(fcs_file_path_list, ignore_channels, dyes,
+     detectors, signal_type, instrument_name, bounds = bounds,
+     minimum_useful_peaks = minimum_fcs_files, max_iterations = max_iterations)
```

The results of the `fit_led` function is a list with the following components:

- `peak_stats` – A list with summary stats for each of the channels for all the different peaks (represented by different FCS files). For each of the channels, peak stats are captured by a data frame with rows corresponding to the different peaks (FCS files) and the following columns:
  - `N`: the number of events in the peak
  - `M`: the mean expression value for that peak
  - `SD`: the standard deviation for that peak
  - `V`: the variance for that peak (square of standard deviation)
  - `W`: the weight of that peak
  - `Omit`: was the peak omitted from the fitting? (TRUE or FALSE)
  - `QR`: the residuals of the quadratic fitting
  - `LR`: the residuals of the linear fitting
  - `QR-I`: the residuals of the iterative quadratic fitting
  - `LR-I`: the residuals of the iterative linear fitting

  The values of `W`, `QR`, `LR`, `QR-I` and `LR-I` will be `NA` if `Omit` is `TRUE`.

  ```
  > ## We have stats for these channels
  > names(led_results$peak_stats)
  > ## Explore the peak stats for a randomly chosen channel (PE-Cy7-A)
  > ## Showing only the head in order to limit the output for the vignette
  > head(led_results$peak_stats$`PE-Cy7-A`)
  ```

- `bg_stats` – A data frame with background stats for each channel. These stats are a convenience way to look at peak stats of the lowest peak. The columns of the data frame correspond to the different channels, and there are the following rows:
  - `N`: the number of events in the lowest peak
  - `M`: the mean expression value for the lowest peak
  - `SD`: the standard deviation for the lowest peak
- `dye_bg_stats` – A data frame with background stats for each "dye". These are the same stats as `bg_stats` described above, but the columns will only be restricted to the detectors/dyes that were listed as arguments of the `fit_led` call. The column headings are converted from channel names (detectors) to "dyes" as per the mapping supplied by the detectors and dyes arguments. The rows of the data frame are the same as with the `bg_stats` described above.

```
> ## Explore bg_stats
> led_results$bg_stats
> ## Explore dye_bg_stats
> led_results$dye_bg_stats
```

- `fits` – For the LED analysis, results are reported for both quadratic fitting and linear fitting (effectively fixing $CV_0 = 0$). Since the uniformity of LED signal outputs is likely to be better than the ability of the cytometer electronics to evaluate them, the c_2 term in a quadratic fit should be very close to zero with a small standard error. If the results of the quadratic fit are consistent with $CV_0 = 0$, we can rely on the linear fit results whose standard errors on c_1 will generally be smaller than the c_1 standard errors of the quadratic fit.

  The `fits` item contains a data frame with fits for each of the channels. The columns of the data frame correspond to the different channels. The rows of the data frame capture the coefficients of both, quadratic fitting and linear fitting as follows:

  - `c0`: value of the c_0 coefficient of the quadratic fitting
  - `c0 SE`: standard error of the c_0 coefficient of the quadratic fitting
  - `c0 P`: the P-value for the c_0 coefficient of the quadratic fitting
  - `c1`: value of the c_1 coefficient of the quadratic fitting
  - `c1 SE`: standard error of the c_1 coefficient of the quadratic fitting
  - `c1 P`: the P-value for the c_1 coefficient of the quadratic fitting
  - `c2`: value of the c_2 coefficient of the quadratic fitting
  - `c2 SE`: standard error of the c_2 coefficient of the quadratic fitting
  - `c2 P`: the P-value for the c_2 coefficient of the quadratic fitting
  - `c0'`: value of the c_0 coefficient of the linear fitting
  - `c0' SE`: standard error of the c_0 coefficient of the linear fitting
  - `c0' P`: the P-value for the c_0 coefficient of the linear fitting
  - `c1'`: value of the c_1 coefficient of the linear fitting
  - `c1' SE`: standard error of the c_1 coefficient of the linear fitting
  - `c1' P`: the P-value for the c_1 coefficient of the linear fitting

- `dye_fits` – A data frame with fits for each "dye". These are the same stats as `fits` described above, but the columns will only be restricted to the detectors/dyes that were listed as arguments of the `fit_led` call. The column headings are converted from channel names (detectors) to "dyes" as per the mapping supplied by the detectors and dyes arguments. The rows of the data frame are the same as with the `fits` described above.

- `iterated_fits` – A data frame with fits for each of the channels in the same way as for the `fits` described above, but based on iterative fitting with weights adjustments.

- `iterated_dye_fits` – A data frame with fits for each of the "dyes" in the same way as for the `dye_fits` described above, ut based on iterative fitting with weights adjustments.

```
> ## Explore dye_fits
> ## fits are the same rows but columns corresponding to all channels
> led_results$dye_fits
> ## Explore iterated_dye_fits
> ## iterated_fits are the same rows but columns corresponding to all channels
> led_results$iterated_dye_fits
```

The Q, B and intrinsic CV_0 can be extracted from the fits using the equations shown in the introduction. For your convenience, this is implemented in the `qb_from_fits` function as shown below in this vignette.

- `iteration_numbers` – A data frame with rows corresponding to all the different channels and 2 columns, Q and L, showing the number of iterations used for the quadratic and linear fitting, resp. This data frame can be reviewed in order to make sure the fitting is converging fast enough and the `max_iterations` parameter is set large enough.

```
> ## Explore iteration numbers
> ## Showing only the head in order to limit the output for the vignette
> head(led_results$iteration_numbers)
```

6 Fitting bead data

The `fit_beads` function performs quadratic fitting for multi-level, multi-dye bead sets. In addition, the `fit_spherotech` function performs fitting for the Sph8 particle sets from Spherotech, and the `fit_thermo_fisher` function performs fitting for the 6-level (TF6) Thermo Fisher set. Internally, this is the same `fit_beads` function except that the number of expected peaks is predefined to 8 and 6, resp.

The parameters for the bead data fitting functions are similar to those required for the LED fitting. The main difference is that a single FCS file (supplied by the `fcs_file_path` argument) is expected because the bead sets are provided as a mixture of the different populations and therefore, acquiring data from a single sample will naturally result in all the peaks contained within a single FCS file. All the beads are expected to have the same (or very similar) light scatter properties. Therefore, we perform automated gating on the forward and side scatter channels in order to isolate the main population. In order to do that, the method requires a `scatter_channels` argument that specifies which 2 channels shall be used for the scatter gating. After the main population is isolated, we use K-means clustering to separate the expression peaks generated by different beads. The number of clusters is pre-defined as 8 for the `fit_spherotech` function, 6 for the `fit_thermo_fisher` function, and provided by the user in the form of the `N_peaks` argument in case of the `fit_beads` function. This clustering is performed on data transformed with the Logicle transformation **??**. Generally, the Logicle *width* (w parameter) of 1.0 has been working well for all our data, but users can change the default by providing a different `logicle_width` value. The rest of the arguments is the same as with LED fitting; the complete list of arguments is a follows:

- `fcs_file_path` – A character string specifying the file path to the FCS file with the acquired bead data.

- `scatter_channels` – A vector of 2 short channel names (values of the $PnN keywords) specifying the 2 channels that should not be used to gate the main bead population. The first channel should be a forward scatter channel, the second one should be a side scatter channel.

- `ignore_channels` – A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

- `N_peaks` – The number of peaks (different beads) to look for. This argument is applicable to the `fit_beads` function only; the `fit_spherotech` and `fit_thermo_fisher` functions have the number of peaks predefined to 8 and 6, resp.

- `dyes` – A vector of dye names. This value does not affect the fitting, but those dyes will be "highlighted" in the provided results.

- `detectors` – A vector of short channel names (values of the $PnN keywords) specifying channels matching to the dyes specified above. The length of this vector shall correspond to the length of the dyes vector. These channels should be all of the same type as specified by the `signal_type` below, i.e., area, height or width of the measured signal.

- `signal_type` – The type of the signal specified as the "area", "height" or "width". This should match to the signal type that is being captured by the channels specified in the detectors argument. The signal type is being used in order to trigger type-specific peak validity checks. Currently, if signal type equals to "height" then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. In addition, peaks that are not sufficiently narrow (*i.e.*, exceeding a specific maximum CV) are also omitted from the fitting. Currently, the maximum allowed CV is set to 0.65, but the code is designed to make this user-configurable and signal type dependent eventually.

- `instrument_name` – The make/model of the instrument. The purpose if this argument is to allow for instrument-specific peak validity checks. At this point, if "BD Accuri" is passed as the instrument type, then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. Additional instrument-specific peak validity checks may be implemented in the future.

- `bounds` – On some instruments, the lowest LED peaks may be cut off at a data baseline so that the peak statistics will not be valid. Therefore, peaks too close to the baseline need to be excluded from the fitting. Also, many instruments do not maintain good linearity to the full top of scale, so it is also important to specify a maximum level for good linearity and, on each fluorescence channel, exclude any peak that is above that maximum. The bounds argument shall provide a list specifying the minimum and maximum value for the means of valid peaks; peaks with means outsize of this range will be ignored for that particular channel.

- `minimum_useful_peaks` – Different peaks may be omitted for different channels due to various validity checks described above. This argument specifies the minimal number of

valid peaks required in order for the fitting procedure to be performed on a particular fluorescence channel.

- `max_iterations` – The maximum number of iterations for the iterative fitting approach with appropriate weight recalculations.

- `logicle_width` – The data clustering part is performed on data transformed with the Logicle transformation **??**. Generally, the Logicle *width* (w parameter) of 1.0 has been working well for all our data, but users can change the default by providing a different value.

```
> ## Example of fitting bead data based on Sph8 particle sets from Spherotech
> fcs_file_path <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
+     "933745.fcs", package="flowQBData")
> scatter_channels <- c("FSC-A", "SSC-A")
> ## Depending on your hardware and input, this may take a few minutes, mainly
> ## due to the required clustering stage of the algorithm.
> spherotech_results <- fit_spherotech(fcs_file_path, scatter_channels,
+     ignore_channels, dyes, detectors, bounds,
+     signal_type, instrument_name, minimum_useful_peaks = 3,
+     max_iterations = max_iterations, logicle_width = 1.0)
> ## This is the same as if we were running
> ## fit_beads(fcs_file_path, scatter_channels,
> ##     ignore_channels, 8, dyes, detectors, bounds,
> ##     signal_type, instrument_name, minimum_useful_peaks = 3,
> ##     max_iterations = max_iterations, logicle_width = 1.0)
```

Next, let's explore the results of this function. Unlike with the LED data, only the results of quadratic fitting are provided for fitting bead data by the `fit_beads` (and `fit_spherotech`, `fit_thermo_fisher`) functions. This is because linear fitting does not account for the intrinsic CV_0 of the beads and is not appropriate for bead data, which always has a significant non-linear component.

The results of the `fit_beads` (and `fit_spherotech`, Rfunctionfit_thermo_fisher) functions is a list with the following components:

- `transformed_data` – A flowCore's *flowFrame* object capturing the data from the input FCS file after the Logicle transformation. This may be reviewed if one wanted to check that the Logicle transformation was appropriate for the input FCS file.

- `peaks` – A list of flowCore's *flowFrame* objects capturing the different peaks identified in the input FCS file by the K-means clustering algorithm. This may be reviewed if one wanted to check that the clustering algorithm identified the peaks correctly.

- `peak_clusters` – The result of the `kmeans` call. This shows the assignment of cluster numbers to input data points and also provides additional details about the clustering; see the documentation for the kmeans function for additional details.

Note that you may see a warning message saying that
`Warning message:`
`Quick-TRANSfer stage steps exceeded maximum (= 3762700)`
According to kmeans documentation, in some cases, when some of the points (rows of 'x') are extremely close, the algorithm may not converge in the "Quick-Transfer" stage, signaling a warning. Based on our experience, this has not negatively affected the result of the clustering. By default, we use the Hartigan and Wong algorithm. We tried other algorithms as well, but those were significantly slower. We also tried to round up the data as advised in the kmeans documentation, but that has not resolved the warning. Below, we demonstrate one way of a simple visual inspection of the clustering results.

```
> library("flowCore")
> plot(
+     exprs(spherotech_results$transformed_data[,"FITC-A"]),
+     exprs(spherotech_results$transformed_data[,"Pacific Blue-A"]),
+     col=spherotech_results$peak_clusters$cluster, pch='.')
```

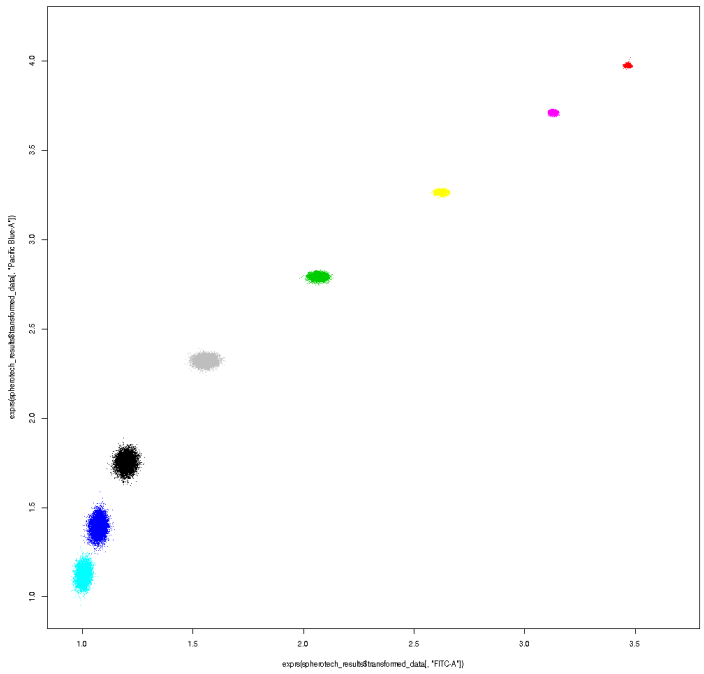

Figure 1: **Result of K-means clustering on bead data.**

- `peak_stats` – A list with summary stats for each of the channels for all the different peaks identified by the K-means clustering algorithm. For each of the channels, peak stats are captured by a data frame with rows corresponding to the different peaks and the following columns:

- `N`: the number of events in the peak
- `M`: the mean expression value for that peak
- `SD`: the standard deviation for that peak
- `V`: the variance for that peak (square of standard deviation)
- `W`: the weight of that peak
- `Omit`: was the peak omitted from the fitting? (TRUE or FALSE)
- `QR`: the residuals of the quadratic fitting
- `QR-I`: the residuals of the iterative quadratic fitting

The values of `W`, `QR`, and `QR-I` will be `NA` if `Omit` is `TRUE`.

```
> ## We have stats for these channels
> names(spherotech_results$peak_stats)
> ## Explore the peak stats for a randomly chosen channel (PE-Cy7-A)
> spherotech_results$peak_stats$`PE-Cy7-A`
```

- `fits` – A a data frame with fits for each of the channels. The columns of the data frame correspond to the different channels. The rows of the data frame capture the coefficients of the quadratic fitting as follows:
  - `c0`: value of the c_0 coefficient
  - `c0 SE`: standard error of the c_0 coefficient
  - `c0 P`: the P-value for the c_0 coefficient
  - `c1`: value of the c_1 coefficient
  - `c1 SE`: standard error of the c_1 coefficient
  - `c1 P`: the P-value for the c_1 coefficient
  - `c2`: value of the c_2 coefficient
  - `c2 SE`: standard error of the c_2 coefficient
  - `c2 P`: the P-value for the c_2 coefficient
- `dye_fits` – A data frame with fits for each "dye". These are the same stats as `fits` described above, but the columns will only be restricted to the detectors/dyes that were listed as arguments of the `fit_beads` (or `fit_spherotech` or Rfunctionfit_thermo_fisher) call. The column headings are converted from channel names (detectors) to "dyes" as per the mapping supplied by the detectors and dyes arguments. The rows of the data frame are the same as with the `fits` described above.
- `iterated_fits` – A data frame with fits for each of the channels in the same way as for the `fits` described above, but based on iterative fitting with weights adjustments.
- `iterated_dye_fits` – A data frame with fits for each of the "dyes" in the same way as for the `dye_fits` described above, ut based on iterative fitting with weights adjustments.

```
> ## Explore fits
> ## Selecting just a few columns to limit the output for the vignette
> spherotech_results$fits[,c(1,3,5,7)]
> ## Explore iterated_fits
> spherotech_results$iterated_fits[,c(1,3,5,7)]
```

The Q, B and intrinsic CV_0 can be extracted from the fits using the equations shown in the introduction. For your convenience, this is implemented in the `qb_from_fits` function as shown below in this vignette.

- `iteration_numbers` – A data frame with rows corresponding to all the different channels and a column, Q, showing the number of iterations used for the quadratic fitting. This data frame can be reviewed in order to make sure the fitting is converging fast enough and the `max_iterations` parameter is set large enough.

```
> ## Explore iteration numbers
> ## Showing only the head in order to limit the output for the vignette
> head(spherotech_results$iteration_numbers)
```

7 Calculating Q and B from fits

The `qb_from_fits` function can be used to calculate Q and B using the equations shown in the introduction. As arguments, the `qb_from_fits` function can take results from both, LED data fitting (`fit_led` function) and bead data fitting (`fit_beads`, `fit_spherotech` function and `fit_thermo_fisher` functions). You can calculate based on any of the fits (`fits`, `dye_fits`, `iterated_fits` and `iterated_dye_fits`).

```
> ## 1 QB from both quadratic and linear fits of LED data
> qb_from_fits(led_results$iterated_dye_fits)
> ## 2 QB from quadratic fitting of bead data
> qb_from_fits(spherotech_results$iterated_dye_fits)
```

The result of this function is a matrix with columns corresponding to the names of the fits used (*e.g.* dyes or detectors). In case of LED fits, the rows will contain the following items:

- `q_QI` The calculated Q_I value of the quadratic fitting.
- `q_BSpe` The calculated B_{Spe} value of the quadratic fitting.
- `q_CV0sq` The calculated CV_0^2 value of the quadratic fitting.
- `l_QI` The calculated Q_I value of the linear fitting.
- `l_BSpe` The calculated B_{Spe} value of the linear fitting.

As expected, you can see that the estimated CV_0^2 values for the LED data are very close to 0 and therefore, the linear model may be preferable. Note that the estimated value can also be small negative since; this is OK since those are just estimates from our model. The true

CV_0^2 should be very close to 0 unless there is something wrong with the instrument or LED data collection.

In case of bead data fits, the rows will contain the following items:

- `q_QI` The calculated Q_I value of the quadratic fitting.
- `q_BSpe` The calculated B_{Spe} value of the quadratic fitting.
- `q_CV0sq` The calculated CV_0^2 value of the quadratic fitting.

8 Parsing house-keeping information from spreadsheets

We designed a simple MS Excel template in order to keep track of FCS files along with the necessary metadata to facilitate fully automated processing of bead and LED data generated by many different instruments. An example of this template is shown in the `inst/extdata/140126_InstEval_Stanford_LSRIIA2.xlsx` file of the flowQBData package. This spreadsheet self-explanatory and captures details, such as the instrument name, location, data folder and file names, dyes or sample names, which channels are fluorescence based and which aren't. Below, we show an example of how the xlsx package can be used in order to process metadata from the spreadsheet and perform the fitting based on that.

```
> ## Example of fitting based on house-keeping information in a spreadsheet
> library(xlsx)
> ## LED Fitting first
> inst_xlsx_path <- system.file("extdata",
+     "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
> xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
> ignore_channels_row <- 9
> ignore_channels <- vector()
> i <- 1
> while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
+     ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
+     i <- i + 1
+ }
> instrument_folder_row <- 9
> instrument_folder_col <- 2
> instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
> folder_column <- 18
> folder_row <- 14
> folder <- xlsx[[folder_column]][[folder_row]]
> fcs_directory <- system.file("extdata", instrument_folder,
+     folder, package="flowQBData")
> fcs_file_path_list <- list.files(fcs_directory, "*.fcs", full.names= TRUE)
> bounds_min_col <- 6
> bounds_min_row <- 7
> bounds_max_col <- 7
```

```
> bounds_max_row <- 7
> bounds <- list()
> if (is.na(xlsx[[bounds_min_col]][[bounds_min_row]])) {
+     bounds$minimum <- -100
+ } else {
+     bounds$minimum <- as.numeric(xlsx[[bounds_min_col]][[bounds_min_row]])
+ }
> if (is.na(xlsx[[bounds_max_col]][[bounds_max_row]])) {
+     bounds$maximum <- 100000
+ } else {
+     bounds$maximum <- as.numeric(xlsx[[bounds_max_col]][[bounds_max_row]])
+ }
> signal_type_col <- 3
> signal_type_row <- 19
> signal_type <- xlsx[[signal_type_col]][[signal_type_row]]
> instrument_name_col <- 2
> instrument_name_row <- 5
> instrument_name <- xlsx[[instrument_name_col]][[instrument_name_row]]
> channel_cols <- 3:12
> dye_row <- 11
> detector_row <- 13
> dyes <- as.character(xlsx[dye_row,channel_cols])
> detectors <- as.character(xlsx[detector_row,channel_cols])
> ## Now we do the LED fitting
> led_results <- fit_led(fcs_file_path_list, ignore_channels, dyes,
+     detectors, signal_type, instrument_name, bounds = bounds,
+     minimum_useful_peaks = 3, max_iterations = 10)
> led_results$iterated_dye_fits
> qb_from_fits(led_results$iterated_dye_fits)
> ## Next we do the bead fitting; this example is with Thermo-fisher beads
> folder_column <- 17
> folder <- xlsx[[folder_column]][[folder_row]]
> filename <- xlsx[[folder_column]][[folder_row+1]]
> fcs_file_path <- system.file("extdata", instrument_folder, folder,
+     filename, package="flowQBData")
> thermo_fisher_results <- fit_thermo_fisher(fcs_file_path, scatter_channels,
+     ignore_channels, dyes, detectors, bounds,
+     signal_type, instrument_name, minimum_useful_peaks = 3,
+     max_iterations = 10, logicle_width = 1.0)
> ## The above is the same as this:
> ## N_peaks <- 6
> ## thermo_fisher_results <- fit_beads(fcs_file_path, scatter_channels,
> ##     ignore_channels, N_peaks, dyes, detectors, bounds,
> ##     signal_type, instrument_name, minimum_useful_peaks = 3,
> ##     max_iterations = 10, logicle_width = 1.0)
```

```
>
> thermo_fisher_results$iterated_dye_fits
> qb_from_fits(thermo_fisher_results$iterated_dye_fits)
```

9 Using data from FlowRepository

We have collected over 900 FCS files from running multi-level beads and LED generated data on over 30 different instruments. This dataset has been uploaded to FlowRepository, it has the public identifier `FR-FCM-ZZTF` and it will be released along with the publication of a paper describing this work. MS Excel spredsheets with all the required metadata to perform the calculations of Q, B and related characteristics are included with that data set. Here, we will demonstrate how to access the data directly from within R in order to perform such calculations.

```
> library("FlowRepositoryR")
> ## 1) Specify your credentials to work with FlowRepository, this will not
> ##    be required once the data is publicly available; see the
> ##    FlowRepositoryR vignette for further details.
> setFlowRepositoryCredentials(email="your@email.com", password="password")
> ##
> ## 2) Get the dataset from FlowRepository
> ##    Note that this does not download the data. You could download all
> ##    data by running
> ##    qbDataset <- download(flowRep.get("FR-FCM-ZZTF"))
> ##    but note that this will download about 3 GB of data
> qbDataset <- flowRep.get("FR-FCM-ZZTF")
> ##
> summary(qbDataset)
> ## A flowRepData object (FlowRepository dataset) Asilomar Instrument
> ## Standardization
> ## 911 FCS files, 36 attachments, NOT downloaded
> ##
> ## 3) See which of the files are MS Excell spredsheets
> spreadsheet_indexes <- which(unlist(lapply(qbDataset@attachments,
+     function(a) { endsWith(a@name, ".xlsx") })))
> ##
> ## 4) Download a random spreadsheet, say the 5th spreadsheet
> ## This is a bit ugly at this point since the data is not public
> ## plus we don't want to download everythink, just one of the files
> library(RCurl)
> h <- getCurlHandle(cookiefile="")
> FlowRepositoryR:::flowRep.login(h)
> ## Once the data is public, only this line and without the curlHandle
> ## argument will be needed:
> qbDataset@attachments[[spreadsheet_indexes[5]]] <- xslx5 <- download(
```

```
+     qbDataset@attachments[[spreadsheet_indexes[5]]], curlHandle=h)
> ## File 140126_InstEval_NIH_Aria_B2.xlsx downloaded.
> ##
> ## 5) Read the spreadsheet
> library(xlsx)
> xlsx <- read.xlsx(xslx5@localpath, 1, headers=FALSE, stringsAsFactors=FALSE)
> ##
> ## 6) Which FCS file contains the Spherotech data?
> ##    This is based on how we create the Excel spreadsheets and
> ##    the FCS file names.
> instrument_row <- 9
> instrument_col <- 2
> instrument_name <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
> fol_col <- 16
> fol_row <- 14
> fol_name <- xlsx[[fol_col]][[fol_row]]
> fcsFilename <- paste(instrument_name, fol_name,
+     xlsx[[fol_col]][[fol_row+1]], sep="_")
> fcsFilename
> ## [1] "NIH Aria_B 140127_Other Tests_BEADS_Spherotech Rainbow1X_012.fcs"
> ##
> ## 7) Let's locate the file in our dataset and let's download it
> ##    Again, the curlHandle is only needed since the file is not public yet
> fcsFileIndex = which(unlist(lapply(qbDataset@fcs.files, function(f) {
+     f@name == fcsFilename })))
> qbDataset@fcs.files[[fcsFileIndex]] <- fcsFile <- download(
+     qbDataset@fcs.files[[fcsFileIndex]], curlHandle=h)
> # File NIH Aria_B 140127_Other Tests_BEADS_Spherotech Rainbow1X_012.fcs
> # downloaded.
> ##
> ## 8) Read in some more metadata from the spreadsheet
> scatter_channels <- c(
+     xlsx[[fol_col]][[fol_row+2]],
+     xlsx[[fol_col]][[fol_row+3]])
> ignore_channels_row <- 9
> ignore_channels <- vector()
> i <- 1
> while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
+     ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
+     i <- i + 1
+ }
> bounds_min_col <- 6
> bounds_min_row <- 7
> bounds_max_col <- 7
> bounds_max_row <- 7
```

```
> bounds <- list()
> if (is.na(xlsx[[bounds_min_col]][[bounds_min_row]])) {
+     bounds$minimum <- -100
+ } else {
+     bounds$minimum <- as.numeric(xlsx[[bounds_min_col]][[bounds_min_row]])
+ }
> if (is.na(xlsx[[bounds_max_col]][[bounds_max_row]])) {
+     bounds$maximum <- 100000
+ } else {
+     bounds$maximum <- as.numeric(xlsx[[bounds_max_col]][[bounds_max_row]])
+ }
> signal_type_col <- 3
> signal_type_row <- 19
> signal_type <- xlsx[[signal_type_col]][[signal_type_row]]
> channel_cols <- 3:12
> dye_row <- 11
> detector_row <- 13
> dyes <- as.character(xlsx[dye_row,channel_cols])
> detectors <- as.character(xlsx[detector_row,channel_cols])
> ##
> ## 9) Let's calculate the fits
> multipeak_results <- fit_spherotech(fcsFile@localpath, scatter_channels,
+     ignore_channels, dyes, detectors, bounds,
+     signal_type, instrument_name)
> ##
> ## 10) And same as before, we can extract Q, B and CV0 from the fits
> qb_from_fits(multipeak_results$iterated_dye_fits)
> #         APC          APC-Cy7      APC-H7       FITC         PE
> # q_QI    0.2406655    0.1291517    0.1291517    0.2034718    0.9014326
> # q_BSpe  34.33642     21.47796     21.47796     21.57055     812.5968
> # q_CV0sq 0.0006431427 0.0004931863 0.0004931863 0.001599552 0.001065658
> #         PE-Cy7       PerCP        PerCP-Cy55   V450         V500-C
> # q_QI    0.2754679    0.08987149   0.08987149   0.07051216   0.192678
> # q_BSpe  11.15762     8.167175     8.167175     24.81647     63.4188
> # q_CV0sq 0.001286111  0.001622208  0.001622208  0.005127177  0.004315592
```

The example shown above processes the Spherotech beads. In order to process the Thermo Fisher beads, one would need to change `fol_col` to 17 in order to determine the proper filename with Thermo Fisher beads results. This is due to how the MS Excel spreadsheet templeta is created. Then, one would use the `fit_thermo_fisher` instead of the `fit_spherotech` function.

If one wanted to process the LED data described by this spredsheet, one would look at columns 3 to 12 to extact the `fol_name` from row 14 and the last part of the filename from row 15. Same as shown in the example above, the instrument name, `fol_name` and the last part of the file name would need to be appended to obtain the complete file name that can be looked up among FCS files in the QB dataset. Specifically, one could lookup the LED FCS files referenced by the spreadsheet as follows:

```
> LEDfcsFileIndexes <- which(unlist(lapply(qbDataset@fcs.files, function(f)
+ {
+     f@name %in% paste(instrument_name, xlsx[14, 3:12], xlsx[15, 3:12], sep="_")
+ })))
> # [1] 173 174 175 176 177 178 179 180 181 182
```

These can than be downloaded and processed analogically to what has been shown above, e.g.,

```
> dir <- tempdir()
> cwd <- getwd()
> setwd(dir)
> lapply(LEDfcsFileIndexes, function(i) {
+     qbDataset@fcs.files[[i]] <- download(qbDataset@fcs.files[[i]],
+         curlHandle=h)
+ })
> setwd(cwd)
> fcs_file_path_list <- list.files(dir, "*.fcs", full.names= TRUE)
> led_results <- fit_led(fcs_file_path_list, ignore_channels, dyes,
+     detectors, signal_type, instrument_name, bounds = bounds)
```

10 Package version

The flowQB package and its functionality have been changed significantly between this and past versions. Functionality described above is only applicable if you have the latest version as shown below.

```
> ## This vignette has been created with the following configuration
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] flowQB_1.18.4

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.6          mvtnorm_1.0-5        lattice_0.20-33
 [4] gWidgets_0.0-54      matrixStats_0.50.2   corpcor_1.6.8
```

```
 [7] digest_0.6.10        rrcov_1.3-11         grid_3.3.1
[10] pcaPP_1.9-60         stats4_3.3.1         gWidgetstcltk_0.0-55
[13] graph_1.50.0         robustbase_0.92-6    flowCore_1.38.2
[16] extremevalues_2.3.2  tools_3.3.1          Biobase_2.32.0
[19] DEoptimR_1.0-6       parallel_3.3.1       BiocGenerics_0.18.0
[22] cluster_2.0.4        tcltk_3.3.1
```